Tesina de Grado
para la obtención del título de
Licenciado en Ciencias de la Computación

Seguridad en iOS y Android: un Análisis Comparativo

Autor
Raúl Ignacio Galuppo
raul.i.galuppo@gmail.com
G-3483/5

Director
Dr. Carlos Luna

Departamento de Ciencias de la Computación
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario
Diciembre de 2017

Índice general

1. Modelo de Seguridad de Android **2**
- 1.1. Entorno aislado para cada aplicación . . . 2
- 1.2. Políticas de acceso mejoradas . . . 3
- 1.3. Seguridad en las aplicaciones . . . 4
  - 1.3.1. Permisos . . . 4
  - 1.3.2. Manifiesto . . . 5

2. Modelo de Seguridad de iOS **6**
- 2.1. Protección de los datos . . . 7
- 2.2. Seguridad en las aplicaciones . . . 7
  - 2.2.1. Entorno seguro . . . 7
  - 2.2.2. Controles de privacidad . . . 8

3. Análisis Comparativo **10**
- 3.1. Analizando Android . . . 10
  - 3.1.1. Autenticación del usuario . . . 10
  - 3.1.2. Seguro desde el arranque . . . 12
  - 3.1.3. Cifrado de la partición de datos . . . 12
  - 3.1.4. Permisos . . . 14
- 3.2. Analizando iOS . . . 16
  - 3.2.1. Bloqueo del dispositivo . . . 16
  - 3.2.2. Arranque seguro . . . 17
  - 3.2.3. Cifrado de archivos . . . 17
  - 3.2.4. Firma de código de las aplicaciones . . . 18
  - 3.2.5. Permisos . . . 18
- 3.3. Crítica . . . 21
  - 3.3.1. Arranque verificado . . . 21
  - 3.3.2. Cifrado del sistema de archivosAAAAAAAAAAAAAAAAAAAAA 21
  - 3.3.3. Bloqueo del dispositivo . . . 22
  - 3.3.4. Seguridad de las aplicaciones . . . 22

4. Una novedosa forma de crear Apps: Apache Cordova **26**
4.0.5. Overview . . . 26
4.0.6. Architecture . . . 26
4.0.7. Plugin Development Guide . . . 27
4.0.8. Native and Hybrid apps – A quick overview . . . 28
4.0.9. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options . . . 29

5. Hacia un Framework Comparativo **31**
5.1. Vista principal . . . 31
5.1.1. Funciones no compatibles . . . 32
5.2. Catálogo de Tests . . . 33
5.2.1. Contactos . . . 33
5.2.2. Calendario . . . 34
5.2.3. Geolocalización . . . 35
5.2.4. SMS . . . 37
5.2.5. Almacenamiento . . . 37
5.2.6. Información del Dispositivo . . . 38
5.2.7. Sensores . . . 39
5.2.8. Internet . . . 40
5.3. Resultados experimentales . . . 41
5.3.1. Clase A . . . 42
5.3.2. Clase B . . . 43
5.3.3. Clase C . . . 43
5.3.4. Clase D . . . 43

6. Conclusiones y Trabajos futuros **45**

Índice de figuras

1.1. Aislamiento de las aplicaciones según su UID [5]. 3
1.2. Los recursos están resguardados por los permisos [4]. 5
1.3. Manifiesto de una aplicación. 5

2.1. Modelo de seguridad de iOS [8]. 6
2.2. Entorno aislado de una aplicación en iOS [9] 8
2.3. Control de privacidad de iOS 9. 9

3.1. Proceso de autenticación en Android[4]. 11
3.2. Diagrama de flujo del *Arranque seguro* [5]. 13
3.3. Captura de `Ajustes/Seguridad`. 14
3.4. Permisos en Android Marsmallow. 16
3.5. Proceso para descifrar un archivo [8]. 18
3.6. Permisos en iOS 9. 19
3.7. *Android App Permissions, API>23* 24

5.1. Áreas del *framework*. 32
5.2. Testeando la administración de los contactos. 34
5.3. Testeando la administración del calendario. 35
5.4. Testeando la geolocalización. 36
5.5. Panel de configuración del emulador de Android. 36
5.6. Testeando los mensajes SMS. 37
5.7. Testeando el almacenamiento del dispositivo. 38
5.8. Testeando Información del Dispositivo. 39
5.9. Testeando los sensores. 40
5.10. Testeando el acceso a Internet. 41

Índice de cuadros

3.1. Comparación de permisos entre Android 6.0 e iOS 23

5.1. Clasificación de permisos según si requieren autorización. . . 42

List of Algorithms

1. Test de Contactos. 33
2. Test del Calendario. 34
3. Test de Geolocalización. 35
4. Test de SMS. 37
5. Test de Almacenamiento. 38
6. Test de Información del Dispositivo. 38
7. Test de los Sensores. 39
8. Test de conexión a Internet. 40

Introducción

Capítulo 1

Modelo de Seguridad de Android

Android [2] es un sistema operativo de código abierto [3], diseñado para dispositivos móviles y desarrollado por Google junto con la Open Handset Alliance [1]. Su arquitectura sigue el estilo arquitectónico conocido como Sistemas Estratificados: los distintos componentes se agrupan en capas según su nivel de abstracción, conformando una jerarquía. Las capas inferiores contienen componentes ligados al *hardware*, mientras que las capas superiores agrupan componentes ligados con tareas de mas alto nivel.
Una de las características principales de Android es que cualquier aplicación, ya sea principal o creada por algún desarrollador, puede, al instalarse con las autorizaciones adecuadas, utilizar tanto los recursos/servicios del dispositivo móvil como los ofrecidos por el resto de las aplicaciones instaladas.
A lo largo de este capitulo se hará una descripción de los principales aspectos del modelo de seguridad de Android, en la version 6.0 llamada Marshmallow, lanzada en 2015.

1.1. Entorno aislado para cada aplicación

Fue una de las primeras tecnologías de seguridad aplicadas en Android, y tiene mucha importancia en el modelo de seguridad. Consiste en que cada aplicación se ejecuta en un *entorno aislado*[1], forzando a que solo pueda tener acceso irrestricto a sus propios recursos. Por lo tanto, las aplicaciones no pueden interactuar entre ellas y tienen acceso limitado al sistema operativo. Cada aplicación se le asigna una única id de usuario UID y se ejecuta en ese usuario como un proceso independiente, tal como se observa en la Figura 1.1.
Dado que el *entorno aislado* es a nivel del *kernel*, este modelo de seguridad

[1]Traducción propuesta para el término *sandbox*.

se extiende al código nativo y a las aplicaciones del sistema operativo, tales como las bibliotecas del sistema operativo y los *frameworks* de las aplicaciones. Esto genera un aislamiento a nivel del *kernel*, ya que todas las políticas que se aplican a usuarios o grupos de usuarios, se transfieren a las aplicaciones por tener su UID.

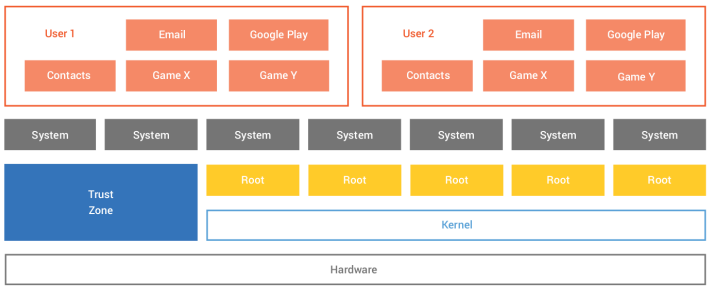


Figura 1.1: Aislamiento de las aplicaciones según su UID [5].

1.2. Políticas de acceso mejoradas

SELinux[2] es un sistema de políticas de acceso obligatorio para Linux. Por lo tanto, siempre se consulta a una autoridad central para permitir cualquier acceso a un recurso; abarca a todos los procesos, inclusive aquellos que corren con privilegios de *root*. *SELinux* opera bajo *the ethos of default denial*, es decir, todo lo que no está explícitamente permitido es denegado. Puede trabajar en dos modos:

- Riguroso: aplica estrictamente las políticas de seguridad.
- Permisivo: no se aplican las políticas pero se guardan en un log.

Decimos que un dominio es una etiqueta que identifica un conjunto de procesos en una política de seguridad. Los procesos que comparten una misma etiqueta de dominio son tratados de la misma forma.
Android permite aplicar el modo permisivo en un determinado dominio y el resto del sistema permanece en modo riguroso. Gracias a ello, se puede lograr aplicar incrementalmente *SELinux* a una porción cada vez mayor del sistema y desarrollar de políticas para nuevos servicios, manteniendo el resto del sistema en vigencia.

[2] *Security Enhanced Linux*, por sus siglas en inglés.

1.3. Seguridad en las aplicaciones

1.3.1. Permisos

Ciertos recursos que provee Android son sensibles, ya que acceden a datos personales o periféricos importantes. Dichos recursos sólo pueden ser accedidos mediante una SS-API[3] con un doble objetivo: tenerlos aislados y permitir cierta granularidad de seguridad sobre ellos [16]. El mecanismo de seguridad para el acceso a estas SS-API de recursos se llama Permisos, tal como se observa en la Figura 1.2.
Podemos clasificar los permisos según el riesgo implícito al otorgarlos, resultando las siguientes cuatro categorías:

- *Normal:* Son aquellos permisos de bajo riesgo ya que corresponden a características aisladas y son considerados de bajo riesgo para las demás aplicaciones, para el sistema y para el usuario. Son concedidos automáticamente por el sistema, sin solicitar aprobación explícita del usuario [18].

- *Dangerous:* Son aquellos permisos de alto riesgo ya que resguardan los accesos a información sensible para el usuario u otorgan control sobre funcionalidades principales del sistema. Para ser concedidos se requiere aprobación explícita del usuario [18].

- *Signature:* Son aquellos permisos que son aprobados solamente si la aplicación que los requiere tiene el mismo certificado que la aplicación que los creó. Cuando el sistema valida el certificado, se otorga el permiso sin requerir aprobación explícita del usuario. Se crearon para permitir que un desarrollador pueda compartir información entre sus distintas aplicaciones sin necesidad de la aprobación del usuario [18].

- *Signature/System:* Son aquellos permisos que controlan el acceso a servicios críticos del sistema. En general, las únicas aplicaciones que los utilizan son las que vienen pre-instaladas en el dispositivo, ya que se utilizan para ciertas situaciones especiales en las que varios proveedores tienen aplicaciones integradas en una imagen del sistema que necesitan compartir funciones específicas explícitamente porque se están creando juntas [18].

[3] *Security Sensitive API*, por sus siglas en inglés.

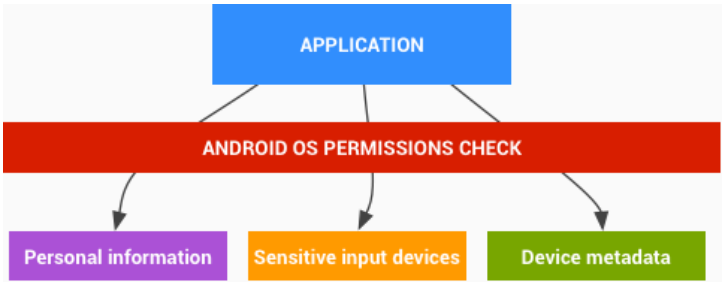


Figura 1.2: Los recursos están resguardados por los permisos [4].

1.3.2. Manifiesto

El archivo de manifiesto proporciona información sobre una aplicación al sistema Android, información que el sistema debe tener para poder ejecutar el código de la aplicación. Es por ello, que todas las aplicaciones deben tener un archivo `AndroidManifest.xml` (con ese nombre exacto) en el directorio raíz. En este archivo se declaran todos los componentes que forman parte de la aplicación en cuestión, los permisos que son requeridos (ver sección 1.3.1) y los permisos exportados por la aplicación, entre otras cosas. En la Figura 1.3 se observa el manifiesto de una aplicación que requiere los permisos `READ_CONTACTS` y `WRITE_CONTACTS`.

```
<?xml version='1.0' encoding='utf-8'?>
<manifest android:hardwareAccelerated="true" android:versionCode="10000"
    android:versionName="1.0.0" package="com.tesina.runtimepermissions"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <supports-screens android:anyDensity="true" android:largeScreens="true"
        android:normalScreens="true" android:resizeable="true"
        android:smallScreens="true" android:xlargeScreens="true" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application android:hardwareAccelerated="true" android:icon="@mipmap/icon"
        android:label="@string/app_name" android:supportsRtl="true">
        <activity android:configChanges="orientation|keyboardHidden|keyboard|
            screenSize|locale" android:label="@string/activity_name"
            android:launchMode="singleTop" android:name="MainActivity"
            android:theme="@android:style/Theme.DeviceDefault.NoActionBar"
            android:windowSoftInputMode="adjustResize">
            <intent-filter android:label="@string/launcher_name">
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name="cordova.plugins.Diagnostic$
            LocationProviderChangedReceiver">
            <intent-filter>
                <action android:name="android.location.PROVIDERS_CHANGED" />
            </intent-filter>
        </receiver>
    </application>
    <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="25" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
</manifest>
```

Figura 1.3: Manifiesto de una aplicación.

Capítulo 2

Modelo de Seguridad de iOS

iOS es un sistema operativo para dispositivos móviles de la multinacional Apple Inc. diseñado para ser seguro [8]. Cada dispositivo combina hardware, software y servicios, diseñados para trabajar conjuntamente para proveer seguridad y al mismo tiempo, que sea transparente para el usuario.
Características como el cifrado del sistema de archivos, asegurar un arranque seguro, proveer un repositorio de contraseñas seguro, vienen habilitadas por defecto. Como se puede observar en la Figura 2.1, la seguridad se extiende más allá del dispositivo, generando un ecosistema seguro.

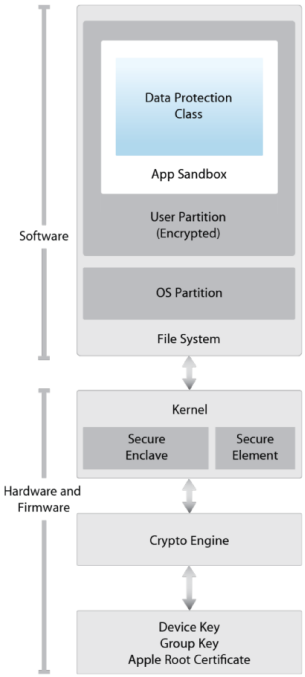


Figura 2.1: Modelo de seguridad de iOS [8].

A lo largo de este capítulo se realizara una descripción de los principales aspectos del modelo de seguridad de iOS, basándose en la versión 9.3, lanzada en 2015.

2.1. Protección de los datos

iOS tiene una protección especial para los archivos y los datos personales, la cual sigue intacta inclusive si algunas otras partes del sistema de seguridad fueron comprometidas [8]. La protección de los datos es implementada construyendo varias claves criptográficas y generando con ellas una jerarquía. El dispositivo móvil provee un componente de *hardware* llamado *Secure Enclave*. Es un coprocesador con memoria cifrada e incluye generación de números aleatorios por hardware. Tiene tres responsabilidades principales:

- proveer las operaciones de cifrado para la manipulación de la Clave de los Datos[1];
- mantener la integridad de dichos datos inclusive si *kernel* haya sido comprometido;
- es el responsable de procesar los datos provenientes del *Touch ID*[2], determinando si se desbloquea el dispositivo.

En los procesadores A9 o posteriores de la serie A, el chip genera de forma segura el identificador único UID[3]. Esta clave fue creada en el momento de fabricación y que no es conocida por Apple ni por ningún otro componente del dispositivo. La misma es utilizada para generar una clave efímera cada vez que se prende el dispositivo, la cual se utiliza para cifrar la memoria del *Secure Enclave* y los datos del sistema de archivos.
El UID permite vincular los datos a un dispositivo determinado mediante cifrado. Por ejemplo, la jerarquía de claves que protege el sistema de archivos incluye el UID, de modo que si los chips de memoria se trasladan físicamente de un dispositivo a otro, no será posible acceder a los archivos.

2.2. Seguridad en las aplicaciones

2.2.1. Entorno seguro

Las aplicaciones son un punto crítico en la seguridad de un dispositivo móvil. Es por ello que iOS provee varias capas de seguridad para las aplicaciones, asegurando que una aplicación esté certificada y verificada

[1] Traducción propuesta para el término *Data Protection Key*.
[2] *Touch ID* es el sistema de detección de huellas digitales que hace posible un acceso seguro, más rápido y sencillo al dispositivo.
[3] *Unique ID*, por sus siglas en inglés.

antes de estar disponibles en la tienda [8].
Cada aplicación instalada por el usuario se ejecuta en un *entorno aislado*[4].

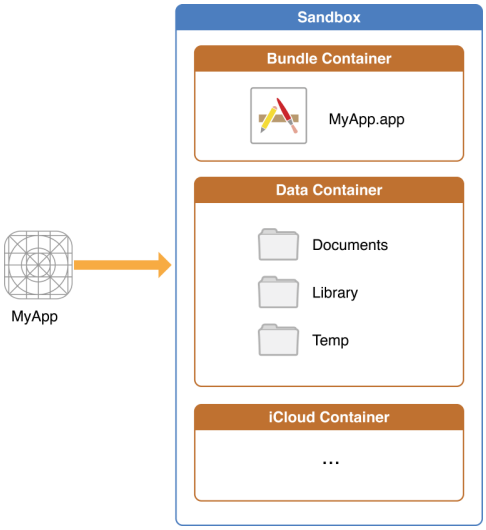


Figura 2.2: Entorno aislado de una aplicación en iOS [9]

Como consecuencia de esto, tiene denegado el acceso de archivos guardados por otra aplicación y no puede realizar cambios en el dispositivo. Si una aplicación requiere acceder a información que no es suya, lo puede hacer únicamente usando servicios de iOS. Lo mismo sucede si quiere ejecutar procesos en segundo plano. En la Figura 2.2 se puede observar lo mencionado anteriormente: cada aplicación tiene su directorio *Home* para sus archivos, el cual es otorgado aleatoriamente al momento de la instalación.
El entorno seguro de una aplicación comienza desde el momento de su desarrollo. Los IDE de iOS construyen las aplicaciones utilizando las técnica ASRL[5]. Por ejemplo, Xcode compila automáticamente con ASLR activada. De esta forma, se asegura que todas las regiones de memoria son aleatorias al momento de ejecución [8], reduciendo la probabilidad de muchos *exploits* sofisticados.

2.2.2. Controles de privacidad

iOS ayuda a evitar que las aplicaciones accedan a la información personal de un usuario sin permiso. Es por ello que, el acceso a ciertos recursos, necesita autorización explícita del usuario. Las aplicaciones pueden solicitar un permiso solamente mientras se esté ejecutando. A su vez, los usuarios pueden optar por no permitir este acceso, y pueden cambiar su elección

[4] Traducción propuesta para el término *sandbox*.
[5] *Address Space Layout Randomization* es una técnica de seguridad informática involucrada en la prevención contra ataques de desbordamiento de *buffer*.

en cualquier momento. Cabe aclarar que una aplicación puede utilizar un recurso sólo si se le ha dado permiso.
Por último, el usuario puede observar cuáles aplicaciones han requerido algún permiso, así como otorgar o revocar cualquier acceso futuro. Toda esta información se encuentra en la configuración de privacidad (`Ajustes/Privacidad`), como se observa en la Figura 2.3.

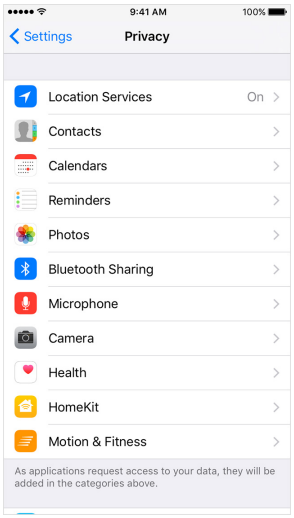


Figura 2.3: Control de privacidad de iOS 9.

Capítulo 3

Análisis Comparativo

Android e iOS son dos plataformas muy populares entre los dispositivos móviles. Es por ello, que existes muchas muchas formas de comparar sus respectivos módulos de seguridad.
La medida de comparación propuesta en [19] consiste en analizar la seguridad de una aplicación móvil en cada fase del ciclo de vida, comparándola en cada plataforma.
En [15] el enfoque es distinto. Se centra en comparar los permisos requeridos a cada plataforma al momento de instalar aplicaciones presentes en ambas plataformas.
En [14, 12, 18] se cambia el enfoque propuesto. El objetivo que persiguen es desarrollar una especificación formal que describa el modulo de seguridad de Android.
En este capítulo se propone una forma de comparar distinta a las anteriores. Consiste en analizar distintas características presentes en ambas plataformas, poniendo foco especialmente en los permisos que se pueden modificar en tiempo de ejecución. El análisis se esta basado en los documentos oficiales de seguridad, tales como [4, 5, 8] entre otros. Al final del capítulo se agrega una crítica sobre las funcionalidades mencionadas en el análisis.

3.1. Analizando Android

3.1.1. Autenticación del usuario

Android provee diversas formas para que un usuario se autentique, con el objetivo de desbloquear la pantalla. Desde los comienzos, la autenticación se realizaba mediante el PIN, contraseña y patrones. A partir de la versión 5.0, se introduce el concepto llamado *TrustAgents*, el cuál permite mecanismos de desbloqueo más flexibles, tales como:

- Reconocimiento facial.
- Un determinado lugar, configurado a través de Google Maps.
- Reconocimiento de voz.
- Ciertos dispositivos, tales como el auto (a través de Bluetooth).

La novedad en la versión 6.0 es que soporta el lector de huellas digitales. Dependiendo del método utilizado para autenticarse, el sistema operativo provee dos componentes: *Gatekeeper* y *Fingerprint*. El primer componente realiza la autenticación del patrón/contraseña del dispositivo en un entorno de ejecución de confianza TEE[1]; mientras que el segundo componente es el encargado de verificar que la huella detectada por el sensor es valida. Ellos interactúan con el *Keystore*[2] para soportar el uso de *tokens* de autenticación respaldados por *hardware* (*AuthTokens*).
La verificación del desbloqueo de la pantalla ocurre en el TEE, tal como se observa en la Figura 3.1. Como consecuencia del esta forma de autenticarse,

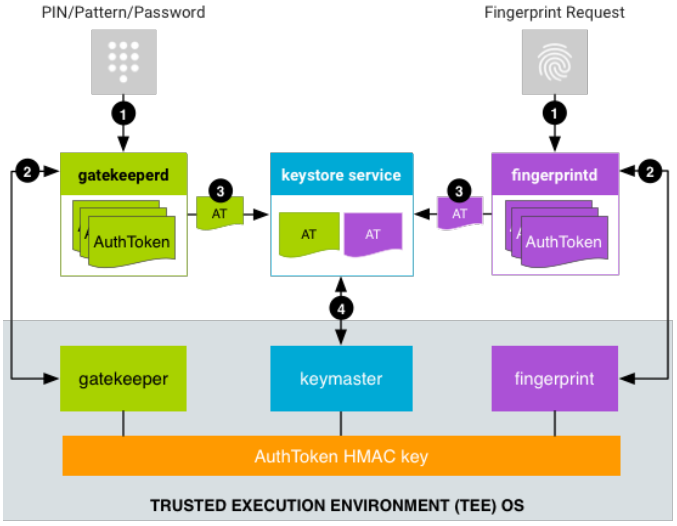


Figura 3.1: Proceso de autenticación en Android[4].

se destacan las siguientes ventajas:

[1]El *Trust Execution Enviroment* es una zona segura del procesador principal en la cual se provee una ejecución segura e íntegra, tanto de código fuente como de datos. El TEE aisla por *hardware* el acceso a cierta memoria y provee mecanismos de I/O para dicha memoria [13].

[2]Es un componente para almacenar las claves criptográficas, el cual dificulta su extracción, ya que asegura dos cosas: una clave nunca entra en una aplicación y una clave nunca sale de una zona segura [7].

- Al permitir el desbloqueo con datos biométricos, se acelera y se simplifica el proceso de autenticación. Los usuarios eligen este sistema de desbloqueo en un 91 % [5].
- Al realizarse la autenticación en un TEE, se mejora la protección contra ataques de fuerza bruta, ya que se incrementa exponencialmente el tiempo de espera para el desbloqueo [5].

Estas mejoras permiten que los desarrolladores de aplicaciones tengan más opciones de seguridad para sus datos y sus comunicaciones.

3.1.2. Seguro desde el arranque

Android ofrece la funcionalidad de garantizar un arranque seguro del dispositivo, comenzando desde un lugar confiable del *hardware* hasta que se monta la partición. Durante el arranque, el sistema operativo verifica que la versión de Android no se haya alterado respecto a la de fábrica, informando mediante alertas en caso contrario y ofreciendo opciones para resolverlo. Dependiendo de la implementación de la funcionalidad, el sistema operativo puede ofrecer una acción al usuario o evitar el arranque hasta que se haya solucionado el problema [4].
En la figura 3.2 se observa el diagrama de flujo del *Arranque Seguro*[3], el cual termina en cuatro estados posibles:

- Verde: indica que se pudo verificar correctamente el arranque del sistema.
- Amarillo: indica que se pudo validar el certificado correspondiente a la partición de arranque. Requiere la huella dactilar para continuar el inicio.
- Naranja: indica que el dispositivo pudo ser modificado, ya que no se pudo verificar la partición de arranque. Requiere acción del usuario para continuar.
- Rojo: indica que falló la verificación. Es decir, no pudo validar ninguna partición.

3.1.3. Cifrado de la partición de datos

Mediante una opción de la configuración, Android permite cifrar todos los datos de usuario presentes en un dispositivo, utilizando claves de cifrado simétricas. Una vez cifrado, el proceso es transparente para el usuario. Es decir, todos los datos creados se cifran antes de enviarlos al disco y todas

[3]Traducción propuesta del término *Verified Boot*.

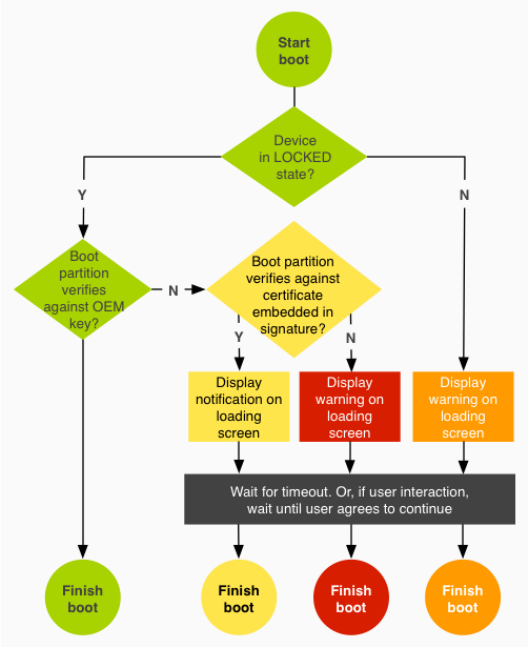


Figura 3.2: Diagrama de flujo del *Arranque seguro* [5].

las lecturas descifran los datos antes de devolverlos al proceso que realizó la llamada.

Esta funcionalidad se activa desde `Ajustes/Seguridad/Cifrar Teléfono`, como se observa en la Figura 3.3. Al activarse dicha opción, se cifran los datos privados de las aplicaciones, el contenido de la tarjeta SD y los datos personales, pudiendo cambiar más adelante el alcance de los componentes afectados por el cifrado. Mientras esté activa dicha opción, cada vez que arranque el dispositivo, el usuario debe proporcionar sus credenciales para poder acceder a cualquier parte del disco.

La primera vez que apareció esta funcionalidad fue en la versión 3.0. Sin embargo, a partir de la versión 5.0, fue fuertemente recomendado a los fabricantes de dispositivos que habiliten esta característica. La novedad incorporada en Android Marshmallow es que se puede cifrar el contenido de la tarjeta SD, permitiendo que sea ilegible si es removida del dispositivo.

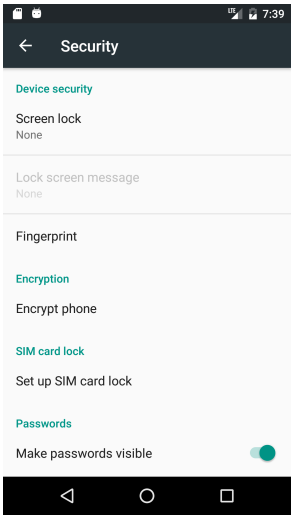


Figura 3.3: Captura de `Ajustes/Seguridad`.

3.1.4. Permisos

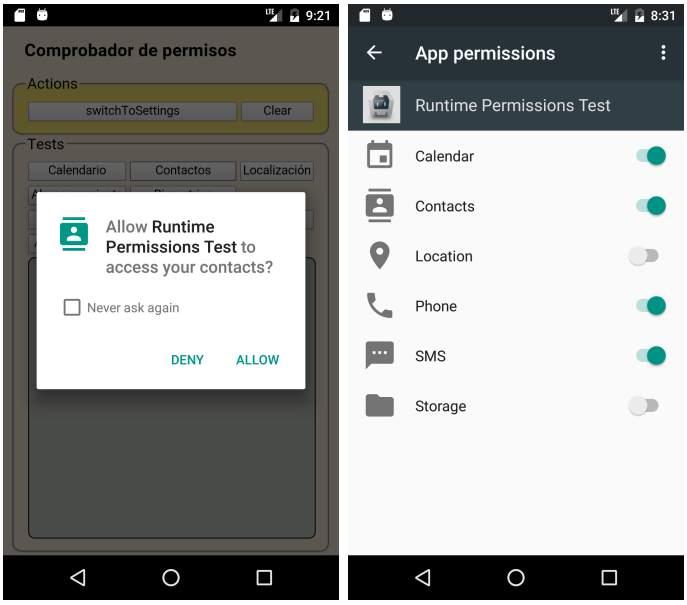
Debido a que cada aplicación Android opera en un *entorno aislado*[4], las aplicaciones deben compartir de manera explícita recursos y datos. El camino utilizado para realizar dicho intercambio es la declaración de permisos, tal como se introdujo en la sección 1.3.1. El presente informe se centra en los permisos *Normales* y *Peligrosos*; cómo se otorgan y cómo se deniegan.
En las versiones anteriores a Android Marshmallow, al prepararse para instalar una aplicación, el sistema operativo mostraba un diálogo al usuario indicando los permisos solicitados y se le solicitaba si deseaba continuar con la instalación. En caso afirmativo, el sistema otorgaba todos los permisos solicitados e instalaba la aplicación. En el caso contrario, no se instalaba la aplicación. El usuario quedaba preso si quería instalar una aplicación: no podía otorgar o denegar permisos individuales; debía otorgar o denegar todos los permisos solicitados como un bloque. Una vez concedidos, los permisos seguían vigentes mientras la aplicación este instalada. Solo se eliminaban si se desinstala dicha aplicación.
A partir de la versión 6.0, se propone un nuevo modelo de permisos, donde los usuarios pueden administrar en tiempo de ejecución los permisos *peligrosos* requeridos por una aplicación. En este modelo, los permisos se agrupan para facilitar el control de la privacidad de los usuarios.
Dichos grupos son:

[4]Ver sección Entorno aislado para cada aplicación.

- *Almacenamiento:* Regula el acceso al almacenamiento externo[5], permitiendo la lectura o la escritura desde el mismo.
- *Calendario:* Permite leer, modificar o eliminar los calendarios del usuario. Incluye además el manejo de los eventos presentes en un calendario.
- *Cámara:* Regula el acceso de la cámara del dispositivo, permitiendo capturar imágenes y grabar videos.
- *Contactos:* Permite leer, modificar o eliminar los contactos presentes en el dispositivo.
- *Localización:* Regula el acceso a la ubicación del dispositivo, ya sea la ubicación precisa (GPS) o la ubicación aproximada (WIFI/Móvil).
- *Mensajes SMS:* Permite escribir mensajes SMS, leerlos o eliminarlos. Además, permite interceptar los mensajes entrantes.
- *Micrófono:* Regula el acceso al micrófono del dispositivo, permitiendo además grabar el sonido obtenido. No se incluyen en este grupo los permisos para capturar sonidos provenientes de una llamada telefónica.
- *Teléfono:* Regula el acceso a la información relacionada a una llamada telefónica, tales como iniciar una llamada, obtener datos de una llamada en curso, manipular el log de llamadas, entre otros.
- *Sensores:* Permite acceder a los datos de los sensores del dispositivo. Se incluyen el giroscopio, el acelerómentro, sensor del ritmo cardíaco, entre otros.

En contraposición a lo que ocurría en las versiones previas, en Android Marsmallow durante la instalación de una aplicación se le otorgan todos los permisos *normales* y ningún permiso *peligroso*. Como consecuencia de esto, cada vez que una aplicación necesita acceder a un recurso protegido por un permiso *peligroso*, tiene que solicitarlo en tiempo de ejecución. Por ejemplo, si una aplicación requiere leer un contacto, la primera vez aparece una notificación pidiendo autorización explicita al usuario, tal como se observa en la Figura 3.4a. El usuario puede otorgar el permiso, denegarlo una vez o denegarlo para siempre. Si elige la última opción, no volverá a aparecer la notificación solicitando dicho permiso. Dichas opciones se encuentran en `Ajustes/Aplicaciones/{nombre de la aplicación}/Permisos`. En la Figura 3.4b se observa una captura de las configuraciones de los permisos para una aplicación móvil.
Debido a que el usuario puede revocar los permisos *peligrosos* en cualquier

[5]En Android, cuando se habla de almacenamiento externo, se refiere a la tarjeta SD.



(a) Solicitud de un permiso en tiempo de ejecución.

(b) Descripción general de los permisos otorgados.

Figura 3.4: Permisos en Android Marsmallow.

momento, una aplicación debe comprobar si dispone de ellos cada vez que se ejecuta. De lo contrario no va a poder acceder a las funciones de las cuales no tiene permiso. Sin embargo, el usuario no puede revocar los permisos *normales*; solamente se eliminan al desinstalar la aplicación.

3.2. Analizando iOS

3.2.1. Bloqueo del dispositivo

Al configurar el código de desbloqueo para un dispositivo, el usuario activa la protección de datos automáticamente. El sistema admite códigos alfanuméricos de cuatro dígitos, de seis dígitos y de longitud arbitraria, salvo que el dispositivo tenga un lector de huellas; en ese último caso deberá contar con al menos seis dígitos.
Además de desbloquear el dispositivo, el código provee entropía a ciertas claves de cifrado del sistema. El hecho de que esté muy ligado con el UID, añade una seguridad extra: no se puede intentar quebrar dicho código fuera del dispositivo. Es por ello que cuanto más seguro sea el código de desbloqueo, más segura será la clave de cifrado.
A fin de desalentar los posibles ataques de fuerza bruta, se generan retardos cada vez mayores tras la introducción de un código inválido en la pantalla de bloqueo. Los retardos están calibrados suponiendo que la frecuencia entre

un ataque y otro es de 80 milisegundos [8]. En dispositivos que cuentan con un *Secure Enclave*, los retardos se aplican mediante dicho componente. Si el dispositivo se reinicia durante un tiempo de demora, la demora aún se aplica, con el temporizador empezando de nuevo para el periodo actual.

3.2.2. Arranque seguro

Cada dispositivo es seguro desde el arranque, ya que la arquitectura del sistema fue pensada para integrar *hardware*, *software* y servicios, con el objetivo de obtener seguridad a lo largo de todos los componentes que conforman el núcleo del sistema [8]. Cuando se prende un dispositivo cuyo sistema operativo es iOS, se siguen los siguientes pasos para asegurar la integridad del arranque del sistema[6] :

1. Se ejecuta el código alojado en la *BootROM*. Dicho código es inmutable y seguro por definición.

2. Se verifica la firma del *bootloader* LLB[7], ya que fue certificado por Apple con la clave pública *Apple Root CA*. La misma está alojada en la *BootROM*.

3. Pasada la validación, empieza la carga del kernel de iOS mediante *iBoot*.

4. El siguiente paso es asegurar la integridad del software. Los dispositivos con un procesador A7 o superior, cuentan con un coprocesador llamado *Secure Enclave*[8] para verificar dicha integridad.

Si fallan alguno de estos pasos, el dispositivo entra en *Modo de Recuperación*[9] y se lo debe conectar a iTunes via USB para restaurar la configuración de fábrica.

3.2.3. Cifrado de archivos

Cada vez que se guarda un archivo en la partición de datos, el sistema operativo crea una clave AES-256 para ése archivo. Dicha clave es única y es utilizada por el *Secure Enclave* para cifrar el archivo. Luego, ésa clave se empaqueta con una (o varias) de las claves de clases, dependiendo de la accesibilidad que va a tener el archivo [8].
Cada vez que se abre un archivo, ocurre el proceso inverso: su metadata

[6] Traducción propuesta para el término *Boot Chain*

[7] *Low Level Bootloader*, por sus siglas en inglés

[8] *Secure Enclave* es un coprocesador con memoria cifrada e incluye generación de números aleatorios por hardware [8].

[9] Traducción del término *recovery mode*.

es desempaquetada con la clave del sistema de archivos, revelando la clave particular del archivo y las clases que lo protegen. Se vuelve a desempaquetar, esta vez, con las claves de las clases, para finalmente descifrar al archivo con su clave única. El proceso completo se observa en la Figura 3.5.

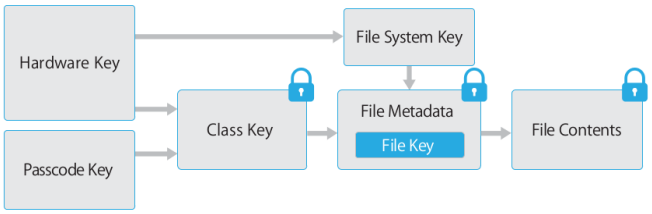


Figura 3.5: Proceso para descifrar un archivo [8].

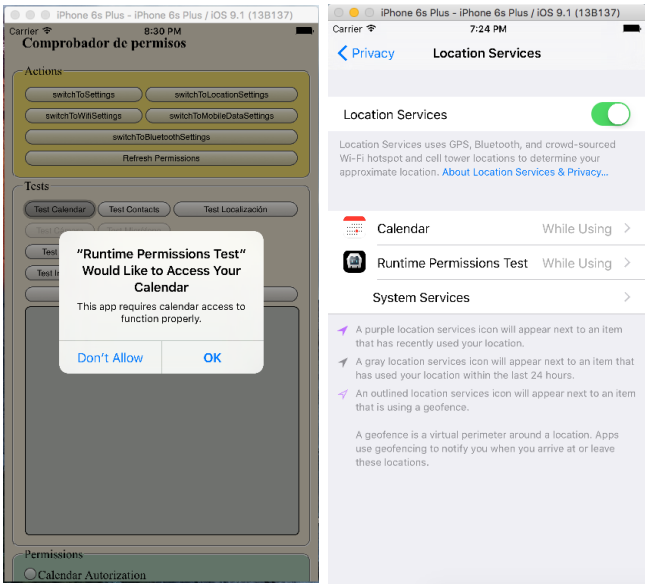
3.2.4. Firma de código de las aplicaciones

iOS garantiza que todas las aplicaciones proceden de una fuente conocida y aprobada. Por ello, requiere que todo el código ejecutable se firme con un certificado emitido por Apple. La firma de código obligatoria extiende el concepto de cadena de confianza del sistema operativo a las aplicaciones e impide que aplicaciones de terceros carguen código sin firmar o utilicen código que se modifique automáticamente.
Para poder desarrollar aplicaciones en dispositivos iOS, los desarrolladores deben registrarse. Apple verifica la identidad real de cada desarrollador, ya sea una persona individual o una empresa, antes de emitir su certificado. Este certificado permite a los desarrolladores firmar sus aplicaciones y enviarlas a la tienda para su distribución. Además, todas las aplicaciones de la tienda han sido revisadas para garantizar que funcionan según lo esperado y que no contienen errores ni otros problemas evidentes.
A diferencia de otras plataformas móviles, iOS no permite a los usuarios instalar aplicaciones procedentes de sitios web que no estén firmadas, ni ejecutar código que no sea de confianza [8]. Durante la ejecución de una aplicación, se comprueba la firma de código de todas las páginas de la memoria ejecutable a medida que se cargan para garantizar que una aplicaciones no se ha modificado desde la última vez que se instaló o actualizó [8].

3.2.5. Permisos

Los controles de privacidad en iOS restringen el acceso de las aplicaciones a la información personal del usuario. El principal control es el sistema de permisos, el cual es encargado de custodiar el acceso a ciertos recursos. Al

momento de instalar una aplicación en el dispositivo, no le se otorga ningún permiso. Es responsabilidad de cada aplicación solicitar los permisos que requiere al momento de utilizar el recurso custodiado. Por ejemplo, en la Figura 3.6a se observa el pedido del permiso al componente Calendario. Sin la autorización explícita del usuario, dicha aplicación tiene restringido el acceso a ese recurso.



(a) Captura del pedido de un permiso.

(b) Aplicaciones que requirieron un permiso.

Figura 3.6: Permisos en iOS 9.

El usuario puede modificar la configuración de privacidad desde `Ajustes/Privacidad`. Se pueden aprobar o denegar cualquier permiso que haya sido solicitado. Aún más, se puede denegar "para siempre": está pensado para evitar que aparezca constantemente una notificación solicitando determinado permiso. En la Figura 3.6b se observan las aplicaciones que requirieron el permiso *Localización*.
Los permisos restringen el acceso a:

- *Localización:* Permite determinar aproximadamente la ubicación del dispositivo. Utiliza una combinación de cierta información del celular, WiFi, Bluetooth y GPS para determinarla.
- *Contactos:* Regula el acceso a los contactos del dispositivo, ya sea para crearlos, modificarlos o eliminarlos.
- *Calendarios:* Regula el acceso a los calendarios, incluyendo las citas y eventos incluidos en él.
- *Recordatorios:* Permite leer, modificar o eliminar un recordatorio. Los recordatorios son pequeñas notas, listas de tareas, entre otras cosas.
- *Fotos:* Regula el acceso a la galería de imágenes de la cámara. También tiene la capacidad de crear un álbum de fotos dentro de la aplicación de fotos.
- *Compartir a través de Bluetooth:* Regula cuáles aplicaciones pueden compartir datos a través de Bluetooth.
- *Micrófono:* Regula el acceso al micrófono.
- *Cámara:* Regula el acceso al la cámara del dispositivo.
- *Salud:* Regula el acceso a la información de salud y estado físico del usuario, tanto las que recopiló el dispositivo como las insertadas por el usuario.
- *HomeKit:* Regula el acceso a los accesorios hogareños registrados en el dispositivo.
- *Redes Sociales:* Permite realizar actividades relacionadas a una red social, tales como crear una sesión de red, obtener el feed de actividad para un usuario, hacer una nueva entrada.
- *Diagnóstico:* Regula cuáles datos de diagnóstico sobre el dispositivo se envían a Apple. Esos datos incluyen información sobre el rendimiento del sistema, capacidad restante en el dispositivo, entre otras.
- *Publicidad:* Permite inhabilitar los anuncios basados en intereses.

3.3. Crítica

Teniendo presente el análisis realizado en las secciones anteriores, se realiza a continuación una crítica sobre los modelos de seguridad de ambas plataformas. La misma está compuesta por cuatro niveles gradualmente más complejos que engloban aspectos de la seguridad que van desde que se prende el dispositvo hasta el uso de una aplicación.

3.3.1. Arranque verificado

Desde la primera versión, iOS ofrece la funcionalidad de verificación desde el arranque del dispositivo para ver si fue modificado respecto de la version de fábrica. Por lo tanto, dicha verificación es transparente para el usuario. En caso de fallar, el dispositivo entra en *modo de recuperación*, teniendo que conectarlo a una computadora para restaurar la configuración de fábrica.
En Android, el arranque verificado se agregó en la version 4.4 pero recién en la versión 6.0 se hizo obligatorio para los fabricantes. La principal ventaja respecto de iOS es que tiene cuatro estados finales de la verificación en vez de dos. Se identifican por colores: verde, amarillo, naranja y rojo. Los estados verde y rojo son equivalentes a los estados de iOS: válido e inválido, respectivamente. Los estados amarillo y naranja corresponden a una validación parcial[10]. Android deja librado a la decisión del usuario para continuar o suspender el arranque.

3.3.2. Cifrado del sistema de archivosAAAAAAAAAAAAAAAAAAAA

Tanto Android con iOS proveen la funcionalidad de cifrar los datos alojados en el sistema de archivos. Sin embargo, tienen una política distinta al momento de aplicarlo: en iOS está siempre habilitada y no se puede deshabilitar; mientras que en Android, se deja esta decisión al usuario.
En Android, al soportar almacenamiento externo, se corre el riego de que se accedan a los datos por fuera del dispositivo. Es por ello que, a partir de la versión 6.0, el usuario puede cifrar la tarjeta SD. De esta manera, reduce el riesgo mencionado.
Por su parte, iOS está libre del riesgo mencionado en el párrafo anterior, ya que todo su almacenamiento es interno. Además, iOS agrega una medida extra de seguridad: los archivos se cifran utilizando varias claves, entre las que se encuentran el código de desbloqueo y UID[11]. Gracias a ello, se dificulta muchísimo la tarea de quebrar el cifrado por fuera del dispositivo.

[10] Ver apartado 3.1.2.
[11] Clave única por dispositivo. Más datos en la sección 2.1.

3.3.3. Bloqueo del dispositivo

Android cuenta con cinco métodos de bloqueo de pantalla: PIN, patrón, contraseña, desbloqueo facial y huella digital. Por otro lado, iOS dispone de sólo dos métodos: PIN y huella digital.
Independientemente del método utilizado para bloquear la pantalla, tanto en Android como en iOS, el proceso de desbloqueo ocurre en un entorno seguro (TEE y *Secure Enclave*, respectivamente). Como consecuencia de esto, las claves se procesan siempre allí y no son conocidas por el resto de los componentes. Además, se incrementa el tiempo de espera del bloqueo, dificultando un ataque por fuerza bruta.
iOS agrega una medida de seguridad extra: el código de desbloqueo está muy ligado al UID. Es por ello que no se puede realizar ataques al código de desbloqueo fuera del dispositivo.

3.3.4. Seguridad de las aplicaciones

Inicialmente, tanto Android como iOS tienen un enfoque similar, en el sentido de que ambos se apoyan en sus propias tiendas de aplicaciones en los que comprueban la seguridad de las miles de aplicaciones que se encuentran a disposición del usuario. Sin embargo, tienen una diferencia filosófica respecto de los proveedores de aplicaciones: en iOS todas las aplicaciones tienen que descargarse de la tienda oficial. Es más, para poder subir una aplicación, su desarrollador pasa por un proceso muy estricto de registro. En cambio, al ser un sistema más abierto, Android favorece la instalación de aplicaciones de terceros, ya que deja abierta la posibilidad de instalar aplicaciones "sueltas" e incluso tiendas. Es peligroso, pero le da una flexibilidad inmensa, frente a la rigidez de iOS. Esta libertad, sin embargo, tiene un precio, y es la presencia de *malware* disfrazado de aplicaciones legítimas.
Respecto a la ejecución de una aplicación, Ambos sistemas aislan los procesos en ejecución en un *entorno aislado*[12]. Como consecuencia de esto, se destacan dos resultados: evita que una aplicación pueda tomar control de todo el sistema; y evita que una aplicación conozca datos de otra.
Al comparar la gestión de permisos de ambas plataformas, encontramos varias similitudes. Lo primera cosa en común es que a las aplicaciones no se le otorgan ningún permiso al momento de instalarla. Otra cosa que comparten es que si una aplicación necesita algún permiso, debe requerirlo mientras se ejecuta, pudiendo el usuario otorgar o denegarlo. La última similitud es que, desde la configuración de privacidad, el usuario puede revocar u otorgar permisos a las aplicaciones.
Respecto a cómo se definen los permisos, se observan diferencias de concepto. En Android están orientados según el riesgo implícito al otorgarlos. En cambio, en iOS, los permisos están orientados a los componentes. Sin

[12]Traducción propuesta para el término *sandbox*.

embargo, se pueden comparar según los componentes que son afectados por un permiso. De esta manera, se puede saber cuáles componentes son protegidos por permisos en ambas plataformas y cuáles están presentes solamente en una de ellas. Con estos datos, se confeccionó la Tabla 3.1.

Permisos		
Ambas plataformas	*Solo en Android*	*Solo en iOS*
Calendario	-	-
Contactos	-	-
Cámara	-	-
Localización	-	-
-	-	Compartir por Bluetooth
Micrófono	-	-
-	Teléfono	-
Sensores	-	-
-	SMS	-
-	Almacenamiento	-
-	-	*Homekit*
-	-	Redes Sociales
-	-	Diagnóstico
-	-	Publicidad

Cuadro 3.1: Comparación de permisos entre Android 6.0 e iOS

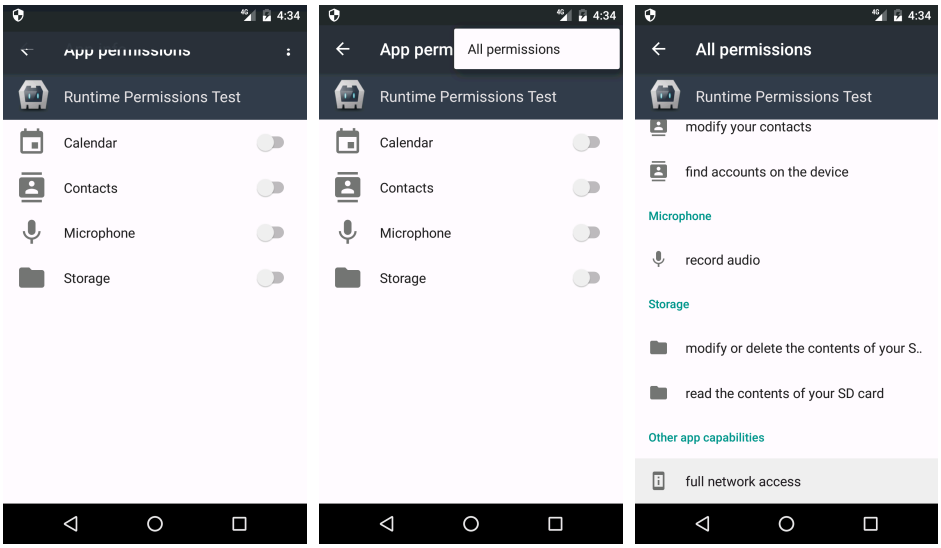
Entre las primeras, podemos mencionar que las dos tienen dispositivos en común a los cuáles se les pueden modificar permisos en *runtime*: micrófono, cámara, GPS, y sensores biométricos. También se aplican dichos permisos a componentes que contienen información sensible del usuario, tales como el calendario, los contactos y las fotos.
Entre las segundas, podemos mencionar permisos que tiene uno pero no el otro. Por ejemplo, Android tiene la permisos sobre el almacenamiento externo. Este permiso no tiene sentido en iOS, ya que los iPhone solamente tienen almacenamiento interno. iOS, por su parte, tiene integrado el manejo de ciertos dispositivos hogareños. También dispone de un servicio tracking de publicidad, el cual permite recomendaciones sobre los gustos del usuario. Por ello, agrega permisos para dichos componentes, los cuales no existes como componentes en Android.
Como consecuencia de lo mencionado anteriormente, se encontraron permisos que son comparables entre sí y otros que no lo son. Cabe aclarar que ciertos permisos se agruparon. Por ejemplo, en Android, los recordatorios forman parte del calendario; mientras que en iOS están separados. Los resultados se volcaron en la tabla 3.1.

Para finalizar el capítulo, se mencionan las conclusiones que se encontraron.
Es importante destacar la falta de granularidad en algunos aspectos. Android no toma como permiso peligroso los datos de las redes sociales. Tampoco el permiso para compartir cosas por Bluetooth. Por otro lado, iOS no tiene la suficiente granularidad para administrar el acceso a las llamadas telefónicas, su historial, etc. Tampoco contempla los datos móviles ni los SMS. En estos casos, el permiso es a nivel de teléfono y no a nivel de aplicación.

Otra cosa a destacar es que en Android el permiso es a nivel de grupo. Cuando se requiere que el usuario permita cierto permiso para una aplicación, lo que sucede es que la aplicación obtiene todos los permisos del grupo. Por ejemplo, si se pide permisos del grupo Çalendario"se obtienen los permisos tanto para leer y para escribir. Para el caso del calendario puede llegar a ser razonable tener los dos permisos pero para otros casos no lo es. Como consecuencia de ello, el usuario esta delegando a una aplicación demasiados permisos.

Detalle de los tests

Luego de instalar la aplicación, se observa que la misma no tiene ningún permiso:



(a) *Runtime Permissions Test* no tiene ningún permiso *peligroso*

(b) Si se presiona sobre los tres puntos, se acceden a todos los permisos

(c) *Runtime Permissions Test* tiene todos los permisos *normales*

Figura 3.7: *Android App Permissions, API>23*

Sin embargo, si se muestran todos los permisos de la aplicación, notará que se tienen todos los permisos *normales* que requiera. Por ejemplo, en la Figura 3.7c se observa que tenemos permiso de acceso a internet.

Capítulo 4

Una novedosa forma de crear Apps: Apache Cordova

4.0.5. Overview

Apache Cordova is an open-source mobile development framework. It allows you to use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's capabilities such as sensors, data, network status, etc. Apache Cordova (formerly PhoneGap) is a mobile application development framework originally created by Nitobi. Adobe Systems purchased Nitobi in 2011, rebranded it as PhoneGap, and later released an open source version of the software called Apache Cordova.[3] Apache Cordova enables software programmers to build applications for mobile devices using CSS3, HTML5, and JavaScript instead of relying on platform-specific APIs like those in Android, iOS, or Windows Phone.[4] It enables wrapping up of CSS, HTML, and JavaScript code depending upon the platform of the device. It extends the features of HTML and JavaScript to work with the device. The resulting applications are hybrid, meaning that they are neither truly native mobile application (because all layout rendering is done via Web views instead of the platform's native UI framework) nor purely Web-based (because they are not just Web apps, but are packaged as apps for distribution and have access to native device APIs). Mixing native and hybrid code snippets has been possible since version 1.9.

4.0.6. Architecture

There are several components to a cordova application. The following diagram shows a high-level view of the cordova application architecture. https://cordova.apache.org/static/img/guide/cordovaapparchitecture.png WebView The Cordova-enabled WebView may provide the application with

its entire user interface. On some platforms, it can also be a component within a larger, hybrid application that mixes the WebView with native application components. (See Embedding WebViews for details.)

Web App This is the part where your application code resides. The application itself is implemented as a web page, by default a local file named index.html, that references CSS, JavaScript, images, media files, or other resources are necessary for it to run. The app executes in a WebView within the native application wrapper, which you distribute to app stores.

This container has a very crucial file - config.xml file that provides information about the app and specifies parameters affecting how it works, such as whether it responds to orientation shifts.

Plugins Plugins are an integral part of the cordova ecosystem. They provide an interface for Cordova and native components to communicate with each other and bindings to standard device APIs. This enables you to invoke native code from JavaScript.

Apache Cordova project maintains a set of plugins called the Core Plugins. These core plugins provide your application to access device capabilities such as battery, camera, contacts, etc.

In addition to the core plugins, there are several third-party plugins which provide additional bindings to features not necessarily available on all platforms. You can search for Cordova plugins using plugin search or npm. You can also develop your own plugins, as described in the Plugin Development Guide. Plugins may be necessary, for example, to communicate between Cordova and custom native components.

NOTE: When you create a Cordova project it does not have any plugins present. This is the new default behavior. Any plugins you desire, even the core plugins, must be explicitly added.

Cordova does not provide any UI widgets or MV* frameworks. Cordova provides only the runtime in which those can execute. If you wish to use UI widgets and/or an MV* framework, you will need to select those and include them in your application.

4.0.7. Plugin Development Guide

A plugin is a package of injected code that allows the Cordova webview within which the app renders to communicate with the native platform on which it runs. Plugins provide access to device and platform functionality that is ordinarily unavailable to web-based apps. All the main Cordova API features are implemented as plugins, and many others are available that enable features such as bar code scanners, NFC communication, or to tailor calendar interfaces. You can search for available plugins on Cordova Plugin Search page.

Plugins comprise a single JavaScript interface along with corresponding native code libraries for each supported platform. In essence this hides the

various native code implementations behind a common JavaScript interface.

This section steps through a simple echo plugin that passes a string from JavaScript to the native platform and back, one that you can use as a model to build far more complex features. This section discusses the basic plugin structure and the outward-facing JavaScript interface. For each corresponding native interface, see the list at the end of this section. https://cordova.apache.org/plugins/ What is a Cordova plugin? A plugin is a bit of add-on code that provides JavaScript interface to native components. They allow your app to use native device capabilities beyond what is available to pure web apps.

4.0.8. Native and Hybrid apps – A quick overview

A native app is a smartphone application developed specifically for a mobile operating system (think Objective-C or Swift for iOS vs. Java for Android).

Since the app is developed within a mature ecosystem following the technical and user experience guidelines of the OS (e.g. swipes, app defined gestures, left aligned header on Android, centrally aligned header on iOS, etcetera), it not only has the advantage of faster performance but also "feels right". What feeling right means is that the in-app interaction has a look and feel consistent with most of the other native apps on the device. The end user is thus more likely to learn how to navigate and use the app faster. Finally, native applications have the significant advantage of being able to easily access and utilize the built-in capabilities of the user's device (e.g., GPS, address book, camera, etcetera). When a user sends text messages, takes pictures using the device's default app, set reminders, or uses the device's music app (the one that came with the phone), they're using native apps.

In short, native apps are exactly that, native to the user's OS and hence built per those guidelines.

Hybrid applications are, at core, websites packaged into a native wrapper.

They look and feel like a native app, but ultimately outside of the basic frame of the application (typically restricted to the controls/navigational elements) they are fueled by a company's website. Basically, a hybrid app is a web app built using HTML5 and JavaScript, wrapped in a native container which loads most of the information on the page as the user navigates through the application (Native apps instead download most of the content when the user first installs the app). Usual suspects here are Facebook, Twitter, Instagram, your mobile banking app, etcetera.

4.0.9. Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options

Currently, the Salesforce Mobile SDK supports building three types of apps: Native apps are specific to a given mobile platform (iOS or Android) using the development tools and language that the respective platform supports (e.g., Xcode and Objective-C with iOS, Eclipse and Java with Android). Native apps look and perform the best. HTML5 apps use standard web technologies—typically HTML5, JavaScript and CSS. This write-once-run-anywhere approach to mobile development creates cross-platform mobile applications that work on multiple devices. While developers can create sophisticated apps with HTML5 and JavaScript alone, some vital limitations remain at the time of this writing, specifically session management, secure offline storage, and access to native device functionality (camera, calendar, geolocation, etc.) Hybrid apps make it possible to embed HTML5 apps inside a thin native container, combining the best (and worst) elements of native and HTML5 apps.

Native apps are usually developed using an integrated development environment (IDE). IDEs provide tools for building debugging, project management, version control, and other tools professional developers need. While iOS and Android apps are developed using different IDEs and languages, there's a lot of parity in the development environments, and there's not much reason to delve into the differences. Simply put, you use the tools required by the device.

An HTML5 mobile app is basically a web page, or series of web pages, that are designed to work on a tiny screen. As such, HTML5 apps are device agnostic and can be opened with any modern mobile browser. And because your content is on the web, it's searchable, which can be a huge benefit depending on the app (shopping, for example).An important part of the "write-once-run-anywhere"HTML5 methodology is that distribution and support is much easier than for native apps. Need to make a bug fix or add features? Done and deployed for all users. For a native app, there are longer development and testing cycles, after which the consumer typically must log into a store and download a new version to get the latest fix. In the last year, HTML5 has emerged as a very popular way for building mobile applications. Multiple UI frameworks are available for solving some of the most complex problems that no developer wants to reinvent. iScroll does a phenomenal job of emulating momentum style scrolling. JQuery Mobile and Sencha Touch provide elegant mobile components, with hundreds if not thousands of plugins that offer everything from carousels to super elaborate controls.However, significant limitations, especially for enterprise mobile, are offline storage and security. While you can implement a semblance of offline capability by caching files on the device, it just isn't a very good solution. Although the underlying database might be encrypted, it's not as

well segmented as a native keychain encryption that protects each app with a developer certificate. Also, if a web app with authentication is launched from the desktop, it will require users to enter their credentials every time the app it is sent to the background. This is a lousy experience for the user. In general, implementing even trivial security measures on a native platform can be complex tasks for a mobile Web developer. Therefore, if security is of the utmost importance, it can be the deciding factor on which mobile technology you choose.
Hybrid development combines the best (or worst) of both the native and HTML5 worlds. We define hybrid as a web app, primarily built using HTML5 and JavaScript, that is then wrapped inside a thin native container that provides access to native platform features. PhoneGap is an example of the most popular container for creating hybrid mobile apps. For the most part, hybrid apps provide the best of both worlds. Existing web developers that have become gurus at optimizing JavaScript, pushing CSS to create beautiful layouts, and writing compliant HTML code that works on any platform can now create sophisticated mobile applications that don't sacrifice the cool native capabilities. In certain circumstances, native developers can write plugins for tasks like image processing, but in cases like this, the devil is in the details. On iOS, the embedded web browser or the UIWebView is not identical to the Safari browser. While the differences are minor, they can cause debugging headaches. That's why it pays off to invest in popular frameworks that have addressed all of the limitations.

Capítulo 5

Hacia un Framework Comparativo

Android e iOS permiten cambiar ciertos permisos de una aplicación luego de haberla instalado en el dispositivo. En este contexto, se ha desarrollado un *framework* para determinar empíricamente el alcance de dichos cambios en los sistemas de permisos de ambas plataformas.
El *framework* es una aplicación móvil y está compuesto por varios tests. Cada test pone a prueba a un componente del dispositivo, permitiendo así conocer el alcance de los permisos correspondientes a dicho componente. De esta manera, se busca dejar en evidencia posibles vulnerabilidades presentes en los modelos de seguridad. Adicionalmente, se hace especial enfoque a la relación existente entre la privacidad del usuario y el sistema de permisos, analizando cuál es la cobertura del sistema respecto de los datos sensibles para la privacidad.
En las siguientes secciones se detallarán los distintos tests que componen el *framework*. Además, se mencionarán las conclusiones arribadas luego de correr los tests mencionados anteriormente.

5.1. Vista principal

Al iniciar la aplicación, se observan dos áreas principales: Acciones y Test, como se puede observar en la Figura 5.1.
El primer área contiene un botón para acceder a la configuración de los permisos del dispositivo. Allí, el *tester* puede cambiar manualmente los permisos requeridos por la aplicación. Además, se encuentra un botón para limpiar la consola del *framework*.
El segundo área se subdivide en dos: la parte de los tests y la parte de la consola. Una parte corresponde a los botones de los tests que, al presionarse, ejecutan el respectivo test, mostrando el resultado en la consola. Dicho resultado se imprimirá con tipografía color verde si fue exitoso; en cambio,

Figura 5.1: Áreas del *framework*.

se imprimirá con tipografía color roja de ser fallido.
A continuación se mencionan los componentes que se pueden testear con el *framework*:

- Contactos
- Calendario
- Geolocalización
- SMS
- Sensores
- Almacenamiento
- Información del dispositivo.
- Acceso a Internet

5.1.1. Funciones no compatibles

El emulador oficial de Android es compatible con la mayoría de las funciones de un dispositivo, pero no incluye la posibilidad de virtualizar los siguientes componentes [6]:

- WiFi;
- Bluetooth;
- NFC[1];
- Manipulación de la tarjeta SD;
- Conexión USB;
- Micrófono;
- Cámara

Al no poder manipular la tarjeta SD, no es posible testear las funcionalidades multimedia: no se puede grabar audio, ni video ni sacar fotos.
Por lo tanto, no se agregaron al *framework* tests para los componentes listados anteriormente.

5.2. Catálogo de Tests

En esta sección se listarán todos los test que conforman el *framework*. Para cada uno de ellos se detalla el algoritmo, los plugins de Apache Cordova que se utilizaron para desarrollarlo y una serie de capturas que muestran los casos exitosos y fallidos.
Para acceder al panel de configuraciones, se utilizó el *plugin* cordova.plugins.diagnostic (v. 3.0.4) de Apache Cordova.
De ahora en adelante, cuando se diga 'consola', se refiere a la consola del *framework*.

5.2.1. Contactos

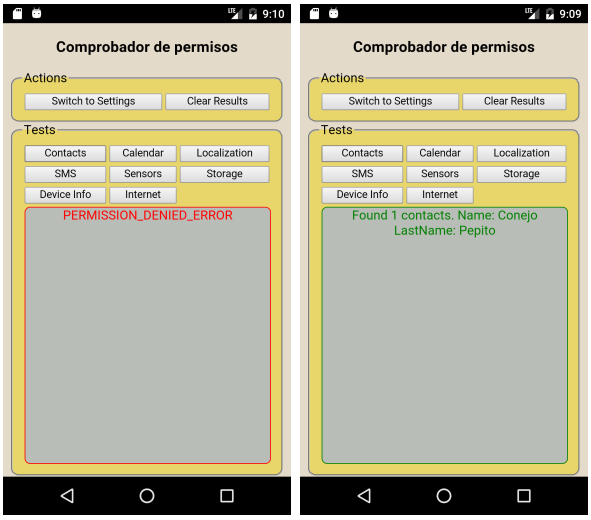
Algorithm 1 Test de Contactos.

1: Se imprime por consola todos los contactos.
2: Se crea un nuevo contacto.
3: Se vuelven a imprimir por consola todos los contactos.

El test consiste en crear un contacto y luego listar todos los contactos presentes en el dispositivo. En caso de ser exitoso, se imprimen los contactos por consola. De lo contrario, se imprime un error. En la Figura 5.2a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 5.2b se observa el caso exitoso.
Para desarrollarlo, se utilizó el *plugin* cordova-plugin-contacts (v. 2.3.1) de

[1]Del ingles *Near Field Communication*. Es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.

Apache Cordova.
Finalmente, para correr el test, es necesario tener el permiso `Contacto`, tanto para Android como para iOS.



(a) Sin permisos. (b) Con permisos.

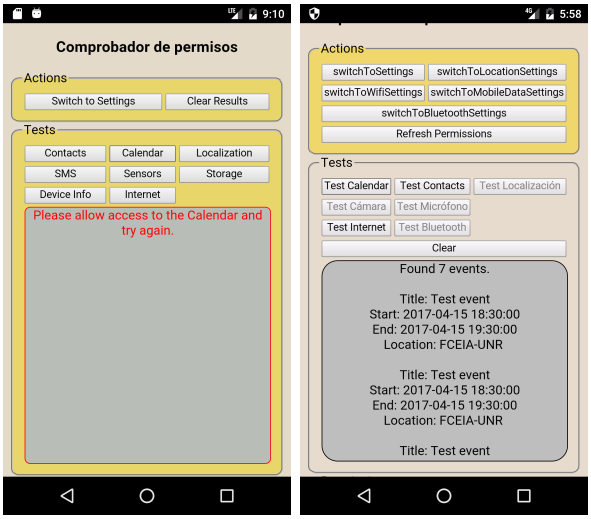
Figura 5.2: Testeando la administración de los contactos.

5.2.2. Calendario

Algorithm 2 Test del Calendario.

1: Se crean las fechas *startDate* y *endDate*.
2: Se crea un evento que empieza en la fecha *startDate* y termina en la fecha *endDate*.
3: Se imprimen por consola los eventos entre las fechas *startDate* y *endDate*.

El test consiste en crear un evento en un determinado rango de fechas y luego listar todos los eventos dentro del rango. En caso de ser exitoso, se muestran los datos del evento. De lo contrario, se muestra un error. En la Figura 5.3a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 5.3b se observa el caso exitoso.
Para desarrollarlo, se utilizó el *plugin* cordova-plugin-calendar (v. 4.6) de Apache Cordova.
Finalmente, para correr el test, es necesario tener el permiso `Calendario` para Android. En cambio, para iOS, es necesario tener el permiso `Recordatorios`.



(a) Sin permisos. (b) Con permisos.

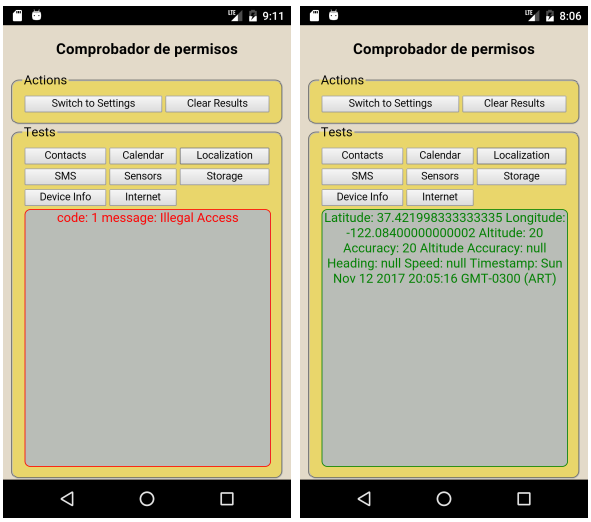
Figura 5.3: Testeando la administración del calendario.

5.2.3. Geolocalización

Algorithm 3 Test de Geolocalización.

1: Se censa el GPS.
2: Se imprimen por consola los datos.

El objetivo del presente test es obtener los datos de la ubicación actual. En caso de tener los permisos correspondientes, obtiene tanto la ubicación precisa (GPS) como la aproximada (WIFI/Móvil). En la Figura 5.4a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 5.4b se observa el caso exitoso.
Para desarrollarlo, se utilizó el *plugin* cordova-plugin-geolocation (v. 2.4.3) de Apache Cordova.
Al momento de realizar las pruebas, se configuró el emulador de Android para que simule las coordenadas `(-122°, 37°)`, tal como se observa en la Figura 5.5. Dicha configuración también se realizo en emulador oficial de iOS.
Finalmente, para correr el test, es necesario tener el permiso `Localización`, tanto para Android como para iOS.



(a) Sin permisos. (b) Con permisos.

Figura 5.4: Testeando la geolocalización.

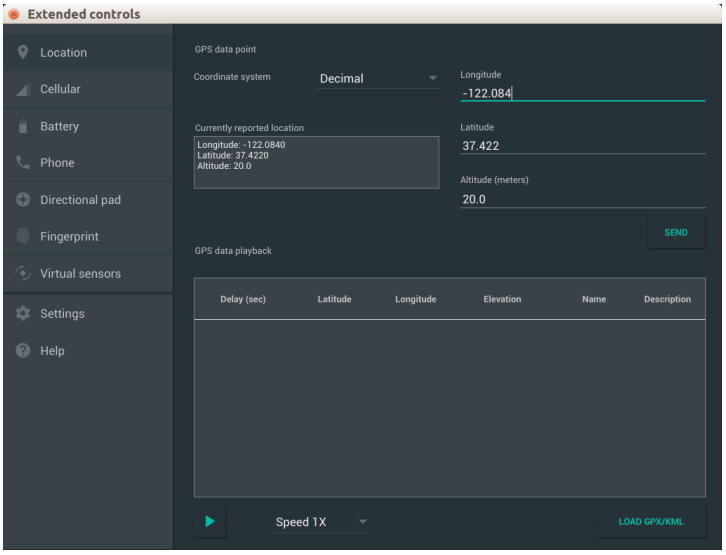


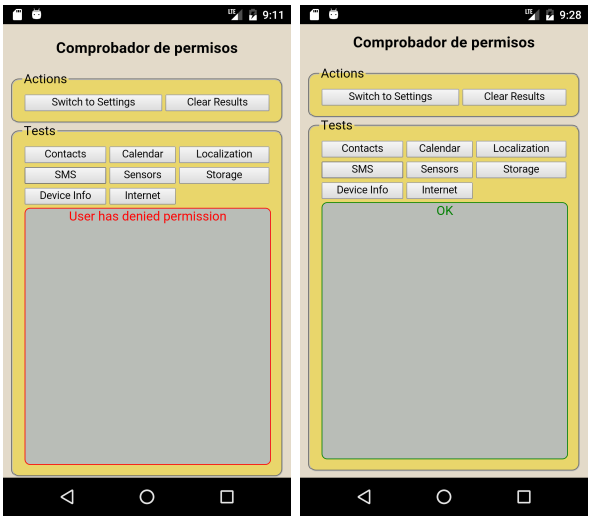
Figura 5.5: Panel de configuración del emulador de Android.

5.2.4. SMS

Algorithm 4 Test de SMS.

1: Se envía un SMS de prueba.
2: Se imprime por consola el resultado del test.

El test consiste en enviar un mensaje SMS. En un principio, se diseñó con la capacidad de enviar, leer y recibir mensajes. Al momento de desarrollarlo, se encontró una restricción de seguridad en iOS: a partir de la version 8 no se pueden acceder a los mensajes SMS desde una aplicación instalada por el usuario [10, 11]. En cambio, en Android sí se pueden acceder, siempre que se tengan el permiso correspondiente. Dicho permiso permiso es `SMS`.
Como consecuencia de lo mencionado en el párrafo anterior, se decidió no implementar las funcionalidades incompatibles, quedando en el test la posibilidad de enviar mensajes SMS. Para desarrollarlo, se utilizó el *plugin* cordova-plugin-sms (v. 0.1.11) de Apache Cordova.
En la Figura 5.6a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 5.6b se observa el caso exitoso.
Finalmente, para correr el test, es necesario tener el permiso `SMS` para Android. Sin embargo, no es necesario tener permisos para correr el test en iOS.



(a) Sin permisos. (b) Con permisos.

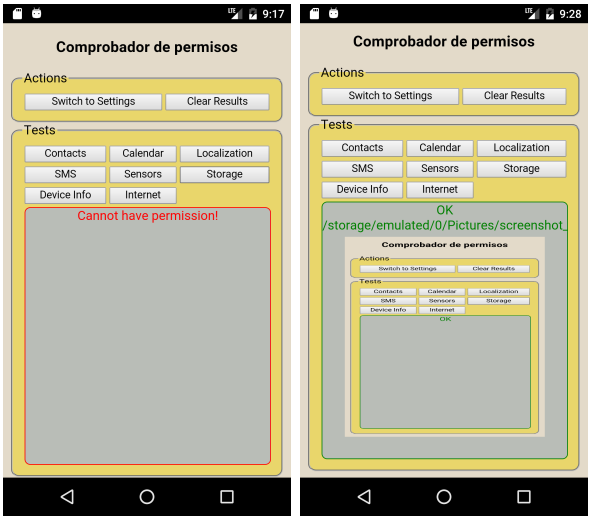
Figura 5.6: Testeando los mensajes SMS.

5.2.5. Almacenamiento

Algorithm 5 Test de Almacenamiento.

1: Se realiza una captura de la pantalla.
2: Se guarda la captura en el dispositivo.
3: Se imprime por consola el resultado del test.

El presente test fue diseñado para probar el alcance de los permisos de escritura sobre el sistema de archivos que tiene cada plataforma. En la Figura 5.7a se observa el resultado del test cuando no se tiene el permiso correspondiente; mientras que en la Figura 5.7b se observa el caso exitoso.
Para desarrollar el presente test se utilizó el *plugin* cordova-screenshot (v. 0.1.5) de Apache Cordova.
Finalmente, para correr el test, es necesario tener el permiso `Almacenamiento` para Android. Sin embargo, no es necesario tener permisos para correr el test en iOS.



(a) Sin permisos. (b) Con permisos.

Figura 5.7: Testeando el almacenamiento del dispositivo.

5.2.6. Información del Dispositivo

Algorithm 6 Test de Información del Dispositivo.

1: Se obtienen los datos del dispositivo.
2: Se imprime por consola los datos obtenidos.

El objetivo de este test es obtener datos del dispositivo donde corre la aplicación. Entre otros datos, se obtiene la plataforma, modelo del

dispositivo y su número de serie. En la Figura 5.8 se observan los datos obtenidos.
Para desarrollarlo se utilizó el *plugin* cordova-plugin-device (v.1.1.6) de Apache Cordova.
Cabe aclarar que no fueron necesarios permisos en ninguna de las dos plataformas para poder correrlo.

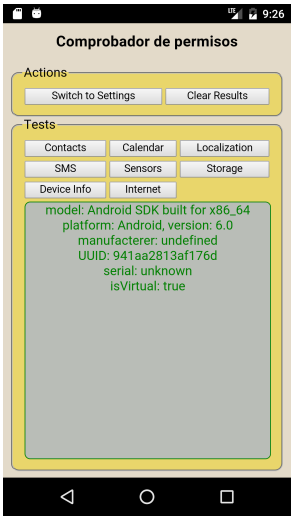


Figura 5.8: Testeando Información del Dispositivo.

5.2.7. Sensores

Algorithm 7 Test de los Sensores.

1: Se inicializa un `timer` con `5 seg` para detener las mediciones.
2: Se inicia la medición del acelerómentro.
3: Se inicia la medición del giroscopio.
4: Se muestran los resultados en la consola.

El objetivo de este test es obtener datos de dos sensores del dispositivo: acelerómentro y giroscopio. Para ello, se configura un `timer`. Durante el tiempo que este activo, se tomarán distintos muestreos; y cuando ocurra el *timeout* se mostrarán los datos en la consola de la aplicación. En la Figura 5.9 se observan los datos obtenidos.
Para desarrollar el presente test se utilizaron los *plugins* cordova-plugin-device-motion y cordova-plugin-gyroscope, de Apache Cordova.
Cabe aclarar que no fueron necesarios permisos en ninguna de las dos plataformas para poder correrlo.

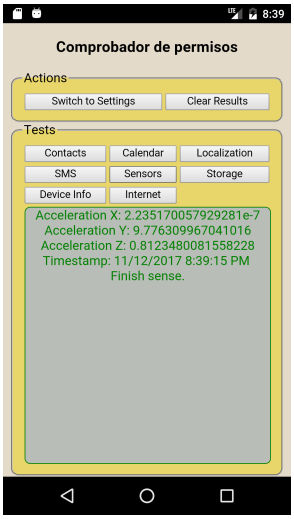


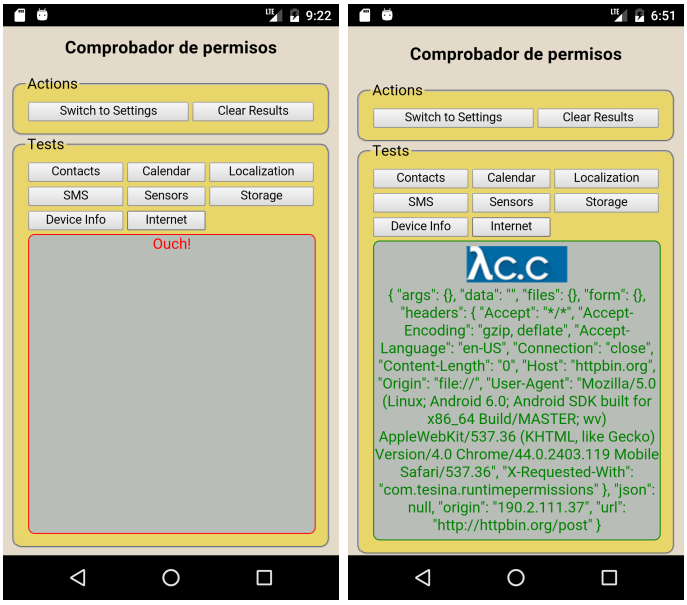
Figura 5.9: Testeando los sensores.

5.2.8. Internet

Algorithm 8 Test de conexión a Internet.

1: Se realiza una consulta GET HTTP hacia logo del DCC.
2: Se decodifica la imagen (viene codificada en Base64).
3: Se imprime por consola un `tag` <IMG>, cuyo `source` es el dato decodificado.
4: Se realiza una consulta POST HTTP hacia httpbin.
5: Se imprime por consola la respuesta.

El objetivo del presente test es establecer una comunicación a través de Internet. Al momento de desarrollarlo, no se requirió de ningún *plugin* de Apache Cordova.
En la Figura 5.10a se observa el resultado del test cuando no se tiene conexión a Internet; mientras que en la Figura 5.10b se observa el caso de establecer una comunicación por Internet.
Cabe aclarar que, al ejecutarse el framework desde un emulador, para probar el acceso a Internet se habilitó/deshabilitó la Red Inalámbrica de la computadora donde se corrieron los emuladores.
Por último, no son necesarios permisos en ninguna de las dos plataformas para poder utilizar el presente test.



(a) Sin conexión a Internet. (b) Con conexión a Internet.

Figura 5.10: Testeando el acceso a Internet.

5.3. Resultados experimentales

El *framework* desarrollado en este trabajo tiene la capacidad de testear los componentes *Contactos*, *Calendario*, *Geolocalización*, *SMS*, *Sensores*, *Almacenamiento*, *Información del dispositivo* y *Acceso a Internet*, tal como se explica en la sección 5.1.
Un primer resultado es poder clasificar los componentes testeados en cuatro clases, según requieran autorización del usuario para utilizarlos. Dichas clases son:

- Clase A: componentes que requieren autorización explícita en ambas plataformas para poder utilizar las funcionalidades que proveen;
- Clase B: componentes que requieren autorización explícita solamente en Android;
- Clase C: componentes que requieren autorización explícita solamente en iOS;
- Clase D: componentes que no requieren autorización explícita para poder utilizar las funcionalidades que proveen.

Las clases son mutuamente excluyentes. Luego de correr los tests provistos por el *framework*, se armó el Cuadro 5.1. Cada test permitió clasificar a un

Permisos			
Clase A	*Clase B*	*Clase C*	*Clase D*
Contactos	-	-	-
Calendario	-	-	-
Geolocalización	-	-	-
-	SMS[2]	-	-
-	Almacenamiento	-	-
-	-	-	Sensores
-	-	-	Información del dispositivo
-	-	-	Acceso a Internet

Cuadro 5.1: Clasificación de permisos según si requieren autorización.

componente presente en ambas plataformas.
A continuación, se detalla como están conformadas las clases.

5.3.1. Clase A

Los componentes que conforman esta clase son *Contactos*, *Calendario* y *Geolocalización*. La característica común entre ellos es que requieren autorización explícita del usuario para interactuar con ellos. Si el usuario deniega el correspondiente permiso, no se puede acceder a ninguna funcionalidad del componente.
El primer componente a analizar es *Contactos*. En Android, requiere el permiso *peligroso* llamado Contacto y en iOS requiere el permiso llamado con el mismo nombre. En ambas plataformas, los dos permisos abarcan las mismas funcionales: permiten crear un contacto, borrarlo, editarlo y obtener un listado de todos los contactos presentes en el dispositivo.
Luego, se continua analizando el componente *Calendario*. En Android permite administrar todo lo relacionado con los eventos: crear un evento, modificarlo y eliminarlo. Además, permite administrar varios calendarios, permitiendo por ejemplo, tener un calendario que contenga los eventos laborales y otro con los eventos personales. En iOS, estas funcionalidades están separadas en dos componentes: *Calendarios* y *Recordatorios*. Cada uno de ellos tiene su permiso que lo regula. Sin embargo, el test presente en el *framework* prueba solamente el componente *Recordatorios*. Para poder correrlo, es necesario tener el permiso Recordatorios. Por otra parte, en Android, el permiso *peligroso* que regula las funcionalidades mencionadas anteriormente, se llama Calendario.
A continuación se analiza el último componente de esta clase, llamado *Geolocalización*. En ambas plataformas, provee el acceso a la ubicación del dispositivo, ya sea la ubicación precisa (GPS) o la aproximada

[2]Aplica solamente al envío de mensajes.

(WIFI/Móvil). Tanto en Android como en iOS, el permiso que regula dichas funcionalidades se llama Localización.

5.3.2. Clase B

Esta clase está compuesta por los componentes *SMS* y *Almacenamiento*. Ellos tienen en común que para utilizar sus funcionalidades no requieren permisos en iOS. Sin embargo, necesitan autorización explícita en Android. Se comenzará el análisis por el componente *SMS*. Sus funcionalidades son: enviar un mensaje, eliminarlo, obtener los mensajes entrantes y obtener la lista de todos los mensajes, ya sean recibidos o enviados. Sin embargo, el test presente en el *framework* solamente prueba el envío de mensajes, ya que en iOS, a partir de la version 8, no permite acceder al resto de las funcionalidades desde una aplicación de terceros (es decir, no nativa) [10, 11]. En cambio, Android permite acceder a todas las funcionalidades mencionadas; y se pueden acceder teniendo el permiso *peligroso* llamado SMS.
Finalmente, se hablará sobre el componente *Almacenamiento*. En Android, el permiso que regula al componente mencionado tiene su mismo nombre. Dicho permiso resguarda las funcionalidades que permiten leer y escribir la tarjeta SD. El test presente en el *framework* solamente prueba escribir en el sistema de archivos.

5.3.3. Clase C

Los componentes que agrupa la presente clase son aquellos que requieren algún permiso en iOS y no requieren permisos en Android para poder utilizar sus funcionalidades.
Cabe aclarar que la presente clase es puramente teórica, ya que ninguno de los componentes analizados pertenecen a ella. Sin embargo, se mantuvo en el informe ya que es posible que exista algún componente en dicha clase, que pueda ser descubierto en futuras versiones del *framework*.

5.3.4. Clase D

La presente clase agrupa componentes que no requieren autorización explícita del usuario para utilizar las funcionalidades que brindan. Sus miembros son *Información del dispositivo*, *Sensores* y *Acceso a Internet*.
Como se explica en la sección 3[3], si una aplicación requiere algún permiso asociado a un componente de esta clase, se le es otorgado por el sistema operativo al momento de instalación. Es por ello que los miembros de esta

[3]TODO: poner la referencia a la sección que explica los permisos en Android

clase pueden ser potencialmente peligrosos respecto de la privacidad del usuario, ya que exponen medios o información que, combinadas a otros componentes, pueden violar dicha privacidad.
Por ejemplo, supongamos que se tiene instalada una aplicación maliciosa de administración de contactos. Supongamos también, para simplificar, que requiere solamente de dos componentes: Contactos y Acceso a Internet. Según el Cuadro 5.1, al usuario se le requeriría explícitamente el permiso *Contactos*. Sin embargo, dicha aplicación maliciosa obtiene el permiso correspondiente a Acceso a Internet sin que el usuario sea notificado. Ergo, ¡puede robar todos los contactos!
Empecemos el análisis de los miembros de la clase. El primer componente nos brinda datos relacionados al dispositivo en el cual se está corriendo la aplicación. Especifica el modelo del dispositivo, el cual es establecido por el fabricante del dispositivo y puede ser diferente en todas las versiones del mismo producto. También se puede obtener información relacionada a la plataforma del dispositivo: nombre, versión, quién lo manufacturó, su UUID[4] y si es emulado o no.
Luego, se continúa con el componente *Sensores*. Dicho componente controla el acceso a los sensores del dispositivo. El test presente en el *framework* recolecta datos de dos sensores: el acelerómentro y el giroscopio. Con los datos obtenidos, se podría calcular varios movimientos del dispositivo, tales como inclinación, vibración, rotación o balanceo.
El último componente de esta clase es el que nos provee acceso a Internet. En la actualidad, son muchas las aplicaciones móviles que utilizan Internet. Según el ranking confeccionado por [17], las 10 aplicaciones más populares utilizan Internet. El test permite realizar una petición GET y una petición POST sin notificar al usuario. Esto es potencialmente muy peligroso ya que una aplicación maliciosa puede enviar y recibir datos sin que el usuario lo note. Por ejemplo, supongamos que tenemos una aplicación maliciosa que nos oficia de navegador GPS. Supongamos también que el usuario le otorga el permiso para utilizar el GPS. Sin embargo, dicha aplicación podría enviar a través de Internet todas las rutas realizadas por el usuario, sin que el mismo sea notificado.

[4]Del ingles Universally Unique Identifier. Es un numero de 128 bits que identifica al dispositivo biunívocamente.

Capítulo 6

Conclusiones y Trabajos futuros

Bibliografía

[1] Alliance, O. H. *Open Handset Alliance.* http://www.openhandsetalliance.com. Último acceso 4 de Noviembre del 2017.

[2] Android. *Android Open Source Project.* https://source.android.com/index.html. Último acceso 4 de Noviembre del 2017.

[3] Android. *Android Open Source Project: Licenses.* https://source.android.com/source/licenses.html. Último acceso 4 de Noviembre del 2017.

[4] Android. *Android Open Source Project: Security.* https://source.android.com/security/index.html. Último acceso 4 de Noviembre del 2017.

[5] Android. *Android Security 2015 Year In Review.* https://source.android.com/security/reports/Google_Android_Security_2015_Report_Final.pdf, 2016. Último acceso 4 de Noviembre del 2017.

[6] Android, D. *Developer Android: Funciones no compatibles.* https://developer.android.com/studio/run/emulator.html#about.

[7] Android, D. *Developer Android: KeyStore System.* https://developer.android.com/training/articles/keystore.html. Último acceso 4 de Noviembre del 2017.

[8] Apple. *Apple iOS: Security Guide.* https://www.apple.com/business/docs/iOS_Security_Guide.pdf, March 2017. Último acceso 4 de Noviembre del 2017.

[9] Apple, D. *iOS Developer Library: File System Programming Guide.* https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html. Último acceso 4 de Noviembre del 2017.

[10] Apple, F. D. *Forum Developer Apple: How to listen for sms reception in ios 8?* https://forums.developer.apple.com/thread/16685. Último acceso 4 de Noviembre del 2017.

[11] Apple, F. D. *Forum Developer Apple: How to read user's sms in an application?* https://forums.developer.apple.com/thread/22447. Último acceso 4 de Noviembre del 2017.

[12] Betarte, G., Campo, J. D., Luna, C. D., and Romano, A. Verifying android's permission model. In *ICTAC 2015* (2015), LNCS 9399, pp. 485–504.

[13] GlobalPlatform. *The Trusted Execution Environment - White Paper*. http://www.globalplatform.org/documents/GlobalPlatform_TEE_White_Paper_Feb2011.pdf, 2011. Último acceso 4 de Noviembre del 2017.

[14] Gorostiaga, F. Especificación e implementación de un prototipo certificado del sistema de permisos de android. Tesina de grado, Licenciatura en Ciencias de la Computación, Universidad Nacional de Rosario, Argentina, Octubre 2016.

[15] Han, J., Yan, Q., Gao, D., Zhou, J., and Deng, H. R. Comparing mobile privacy protection through cross-platform applications. *Network & Distributed System Security Symposium (NDSS)* (2013).

[16] Han, J., Yan, Q., Gao, D., Zhou, J., and Deng, H. R. Android or ios for better privacy protection? *International Conference on Secure Knowledge Mangagement in Big-data era (SKM)* (2014).

[17] of Apps, B. *App Download and Usage Statistics 2017*. http://www.businessofapps.com/data/app-statistics/. Último acceso 4 de Noviembre del 2017.

[18] Romano, A. Descripción y análisis formal del modelo de seguridad de android. Tesina de grado, Licenciatura en Ciencias de la Computación, Universidad Nacional de Rosario, Argentina, Junio 2014.

[19] Yogita Chittoria, N. A. Application security in android-os vs ios. *International Journal of Advanced Research in Computer Science and Software Engineering 4* (2014).